Symcloud

Distributed Filestorage and Colaboration-Platform

Über das Projekt ...

Die Idee zu diesem Projekt kam mir vor etwa einem halben Jahr als wir aufgefordert wurden nach einem Thema für die Master-Arbeit zu suchen. Ich habe mir das Thema ohne konkretes Ziel ausgesucht.

ownCloud

Da ich das Projekt ownCloud schon seit einigen Jahren verfolge und das Projekt genial finde, kam mir die Idee etwas zu diesem Thema zu schreiben. Auch versuchte ich in der Vergangenheit schon ein solches Projekt aufzuziehen, um meine eigenen Daten zu verwalten. Dies scheiterte aber und landeten in der Tonne.

Seit ich bei Massiveart an SULU arbeite, kenne ich die Vorzüge der Programmierung mit dem Symfony-Framework. Da kommt einem die Tatsache nicht wirklich entgegen, das ownCloud aufgrund des alters auf weniger Moderne Art entwickelt wurde. Daher fiel es flach eine Erweiterung oder ähnliches für ownCloud zu schreiben.

Nach kurzer Recherche, wurde mir bewusst, dass ich auf eine nicht gefüllte Nische gestoßen bin. Es gibt keinen Cloud-Storage auf Symfonybasis, mit dem es möglich wäre eine Plattform wie ownCloud zu implementieren.

Punch

Das traf mich wie ein Schlag ...

Spezifikationen

Die Idee einen Storage auf Basis von Symfony zu entwickeln, der es ermöglichte Dateien zu versionieren, teilen, verteilen und sicher zu verwalten, war geboren.

Inspiration Project Xanadu

Nach Recherchen zu diesem Thema fand ich ein hoch Interessantes Projekt aus den 1960er Jahren. Das Projekt Xanadu. Es wurde eben in der 60ern von dem US-Amerikanischen Philosophen Ted Nelson initiiert.

Project Xanadu

Er befasste sich in diesem Projekt mit den Themen Transclusion von Dokumenten und prägte damit den Begriff des Hypertext. Das heutige HTML ist eine Teilmenge seiner Specifikation, die er 1974 und 1981 in den Bücher Dream Machines und Literary Machines veröffentlichte. Das Projekt hatte das Ziel Computernetzwerke mit einer intuitiven Oberfläche zu bauen, die in der Lage sind Relationen zwischen Dokumente effizient zu verwalten und darzustellen.

Ted Nelson arbeitete sein Leben lang daran die Software zu implementieren. In einem Artikel des Wired Magazins wurde es 1995 das "am längsten dauernde Vaporware-Projekt in der Geschichte der EDV" genannt. In einem empörten Brief an die Wired Redaktion wendete Nelson ein, dass einige Aspekte seiner Vision dabei waren, durch Tim Berners-Lees Erfindung des World Wide Webs umgesetzt zu werden.

Obwohl er die Umsetzung von Berners-Lee ablehnte ...

Zitat

„HTML ist exakt was wir zu VERHINDERN versucht haben - ständig tote Links, Links die nur nach außen führen, Zitate, die man nicht zu ihren Ursprüngen zurückverfolgen kann, keine Versionsverwaltung, keine Rechteverwaltung." – Ted Nelson

Das Projekt definierte schon zu Beginn Features wie:

- Transclusion (also die teilweise Einbindung von Objekte in eine anderes) - Zitate
- Bidirektionale Links zwischen den Dokumenten
- Micropayment für Transclusion Zitate oder andere Dokumenten verwertung
- Versionen Dateien sollen versioniert werden und Versionen sollen immer erreichbar bleiben, wenn sie veröffentlicht wurden.
- Redundanzen auf Speicherebene um die Datensicherheit zu erhöhen

Der ursprüngliche Entwurf von Xanadu bestand aus 17 Thesen.

Wichtige Thesen

Einige davon sind relevant für ein Projekt wie Symcloud:

- Every Xanadu server can be operated independently or in a network.
- Every user can search, retrieve, create and store documents.
- Every document can be rapidly searched, stored and retrieved without user knowledge of where it is physically stored.
- Every document is automatically stored redundantly to maintain availability even in case of a disaster.

Diaspora*

Das zweite Projekt, dass eine Inspirationsquelle war das verteilte Soziale Netzwerk Diaspora.

Hard Facts

Es wurde im Jahre 2010 von den vier Mathematikstudenten Dan Grippi, Maxwell Salzberg, Raphael Sofaer und Ilya Zhitomirskiy initiiert und umgesetzt.

Network

Die einzelnen Installationen von Diaspora, Pods genannt, bilden zusammen ein Dezentrales Peer-To-Peer Netzwerk um Datensätze wie Bilder oder Kommentare auszutauschen. Dabei gelten bei Diaspora die Grundsätze decentralization, freedom, privacy => you own your data.

Awesome

Eine Kombination dieser drei Projekte wäre doch awesome nicht?

All in All

Aufgrund dieser Inspirationsquellen, reifte die Idee für die Arbeit immer weiter. Das eigentliche Thema dabei war: Evaluierung und Entwicklung eines Verteilten Speicherkonzeptes als Grundlage für eine Filehosting und Collaboration Platform

Vision

Meine Ideen für das Projekt reichen allerdings noch viel weiter. Durch das einfache Konzept, wäre es möglich jede beliebige Anwendungen miteinander zu verknüpfen, die mit Dateien arbeiten und das Konzept von Symcloud umsetzt. Dabei sollte es nicht auf eine Plattform oder Programmiersprache beschränkt sein. Es sollte mit dem Konzept möglich sein “Alles zu verbinden”.

Raw PHP

Der Start-Schuss dazu gibt die Implementierung von symCloud in PHP. Es ist als Library implementiert und unabhängig von der Applikation in der es eingesetzt wird. Es verwendet zwar Teile des Symfony Frameworks, kann aber in jede beliebige Applikation eingebunden werden, da es keinerlei andere Abhängigkeiten

besitzt. Also wäre es jetzt schon möglich alle PHP Applikationen auf einfachste weise miteinander kommunizieren zu lassen. Das Bootstrap-Script mit silex keine 100 Zeilen lang.

Auch andere Programmiersprachen sind denkbar wie zum Beispiel Go oder NodeJs.

Description

Dafür setzt symCloud auf standardisierte Web-Standards wie zum Beispiel: HTTP, JSON oder REST-Services und auf eine Architektur, die in allen Programmiersprachen entwickelt werden kann.

Zusammenfassend ist symCloud keine einzelne Plattform, es ist eine Spezifikation für eine offene Filehosting-Cloud.

Konzept

Kommen wir nun zu dem Technischen Part und durchleuchten das Konzept etwas genauer.

Übersicht

Bild von Übersicht (Abbildung 10)

Datenmodell

Das Konzept von symCloud wurde rundum das Datenmodell definiert. Das Datenmodell wiederum ist eine Weiterentwicklung des Datenmodells von GIT. Wer dieses Datenmodell nicht kennt, ist hier ein Beispiel dafür.

Bild von GIT-Datenmodell (Abbildung 13)

TODO Beschreibung

- Immutable Objekte

Um dieses Datenmodell fit zu machen wurde ein paar Änderungen durchgeführt.

Bild von symCloud-Datenmodell (Abbildung 11 - vereinfacht)

- Dateien werden nicht komplett gespeichert, sondern werden in sogenannte chunks aufgeteilt.

- Nicht berücksichtigt wurde, im Datenmodel von GIT, die Zuordnung der Referenzen zu einem Benutzer. Diese Zuordnung wird von Symcloud verwendet, um die Zugriffsrechte zu realisieren. Ein Benutzer kann einem anderen Benutzer die Rechte auf eine Referenz übertragen, auf die er Zugriff besitzt. Dadurch können Dateien und Strukturen geteilt und zusammen verwendet werden (Shares).
- Die dritte Erweiterung ist die Verbindung zwischen Tree und Referenz. Diese Verbindung verwendet Symcloud um Symlinks (zu Referenzen) in einem Dateibaum zu modellieren und dadurch die Einbettung von Shares in den Dateibaum zu ermöglichen. Diese Verbindung ist unabhängig von dem aktuellen Commit der Referenz und dadurch ist die gemeinsame Verwendung der Dateien zwischen den Benutzern einfach umzusetzen.
- Die Policies oder Strategien werden verwendet, um zusätzliche Informationen zu den Benutzerrechten bzw. Replikationen in einem Objekt zu speichern. Es beinhaltet im Falle der Replikationen den primary Server bzw. eine Liste von backup Servern, auf denen das Objekt gespeichert wurde.

Dieses Datenmodell unterstützt die Kern-Features:

- Versionierung
- Teilen
- Rechteverwaltung

Das Datenmodell ist also der Zentrale Bestandteil, des Konzeptes. Die Bibliothek, die dieses Modell umsetzt, muss sich darum kümmern, das die Daten verteilt werden und die Konsistenz zwischen den Servern gewahrt wird.

Datenbank

Die Datenbank ist eine eigene entwicklung von Symcloud. Sie ist allerdings nicht sehr aufwendig. Es handelt sich im Grunde genommen um eine einfache Hash(key)-Value Datenbank. Dadurch ist sie vergleichbar mit der Datenbank von GIT. Allerdings wurde die Datenbank so gestalltet, dass sie mit verschiedenen Typen von Daten umgehen kann und nicht an das Datenmodell gebunden ist. Durch events bei den Store und Fetch Vorgängen ist sie flexibel erweiterbar.

Store

Sequencediagramm von Store

Der Speichervorgang ist im Grunde eine Serialisierung anhand von Klassenmetadaten. Anschließend wird das store event geworfen. Eventhandler haben daraufhin die Möglichkeit die Daten zu bearbeiten oder den Vorgang abzubrechen.

Über den StorageAdapter werden die Daten mit dem Hash als Key auf das Speichermedium geschrieben und mit dem SearchAdapter werden die Objektmetadaten indexiert um eine Suche zu ermöglichen.

Welche Daten serialisiert bzw. indexiert werden, kann in den Klassenmetadaten spezifiziert werden. Dabei ünterstützt die Datenbank auch Referenzen, die als Proxies aus der Datenbank geladen werden um Unendliche abhängigkeiten zu vermeiden.

Fetch

Sequencediagramm von Fetch

Der Ladevorgang ist genau das gegenteil. Um das laden von Daten anderer Server zu ermöglichen, wird das fetch event auch geworfen wenn die Daten im Lokalen Storage nicht gefunden werden.

Auch hier haben die Eventhandler wider die Möglichkeit die Daten anzupassen oder abzubrechen.

Diese beiden Vorgänge beschreiben eine einfache lokale Datenbank.

Replicator

- Erstellt Replikationen im Netzwerk
- Kümmert sich um Caching
- Lädt Daten von anderen Servern nach
- Um die Konsistenz zu wahren implementiert symCloud ein einfaches primärbasiertes Protokoll, bei dem jedes Objekt einen Primären und eine gewisse Anzahl von Backup Servern besitzt. Der Primäre-Server ist dabei der Ersteller des Objektes und zuständig für die Auslieferung, Bearbeitung und die Benutzerrechte.

XtreemFS

Als Inspiration für ein solches Protokoll habe ich das verteilte Dateisysmte XtreemFS hergenommen. Es implementiert ebenfalls ein komplexeres Primärbasiertes Protokoll.

Im Grunde besteht das Protokoll aus 5 Punkten:

1. Der Client kontaktiert eine Server
2. Dieser Server sich mit den anderen Servern und übernimmt den Primary Server für die angefragte Datei
3. Der Client kann nun auf diesem Server Änderungen an der Datei ausführen. Für Lesezugriffe ist keine Kommunikation mit den anderen Servern nötig.

4. Schreiboperationen werden zuerst lokal auf dem Primary-Server durchgeführt und dann an die Backup-Server weitergeleitet. Sobald die Backups das ACK zurücksenden, ist der Zustand des Systems wider konsistent
5. Wenn der Primary-Server der Datei ausfällt, kann der Client einen beliebigen anderen Server anfragen.

Symcloud

Für die Symcloud Datenbank habe ich eine einfachere Variante gewählt. Die die Daten immer dort bearbeiten, wo sie engelegt wurden. Dies ermöglicht nicht nur eine einfachere Implementierung es verhindert auch, dass sich bestimmte Server die Kontrolle über Objekte verschaffen, die sie nicht sehen sollten.

Für Symcloud bedeutet das, aufgrund der immuatable Objekte aber kaum Einschränkungen. Nur der Zustand von Referenzen müssen live beim Server des Besitzers angefragt bzw. bearbeitet werden. Da alle anderen Objekte nicht bearbeitet werden.

Wird ein Objekt angelegt, wird dem Objekt eine sogenannte Policy zugewissen, in dem der aktuelle Server als Primary definiert wurde. Anhand von bestimmten Kriterien werden nun die Backup-Server für das Objektermittelt.

Es gibt drei Möglichkeiten, diese zu ermitteln.

- Vollständig: Das Objekt wird gleichmäßig auf eine konfigurierbare Anzahl von Servern verteilt, dabei ist das Objekt offen zugänglich für alle Benutzer
- Rechte: Das Objekt wird aufgrund der Benutzerrechte auf die Server verteilt. Dabei wird es auf den Servern erstellt, auf denen Benutzer Zugriffsrechte auf das Objekt besitzen.
- Stub: Der Typ stub bezeichnet das Anlegen des Objektes auf allen Servern. Dieses Objekt wird aber nicht komplett übertragen, sondern ist im Prinzip nur als Link zu dem Primary-Server zu verstehen. Es enthält also einen Link wo dieses Objekt zu finden ist.

Wenn ein Server, die Referenz eines anderen Benutzers verändern will, erstellt der Server neue Tree und Commit Objekte. Dann versucht er die Referenz zu bearbeiten und macht daher eine “store” Anfrage an den Primäry-Server der Referenz. Der Primary-Server kann diese Anfrage autentifizieren, validieren und ausführen, wenn alle Tests erfolgreich waren.

Wenn es hier zu einem Konflikt kommt, ist nur der eine Server betroffen und dieser kann versuchen den Konflikt aufzulösen. Strategien dazu wurden noch nicht betrachtet.

Es wenn der Server eine korrekte Anfrage stellt, bekommen es die anderen Server mit. Daher ist die Konsistenz des Systems zu jedem Zeitpunkt garantiert.

Ablaufdiagramm Konflikt

Diese einfachen Mechanismen, machen das Konzept übertragbar in fast alle Programmiersprachen. Auch der Speicherlayer ist variabel austauschbar. Neben dem Filesystem sind auch verteilte Datenbanken wie MongoDB, Riak oder Redis denkbar. Dies würde auch die Datensicherheit auf dem einzelnen Server verbessern. Dieser Storagelayer kann aber auch auf jedem Server anders gewählt werden. Dem gesamten System ist es egal wo die Daten liegen.

Weitere Entwicklungen

In einem weiteren Schritt sollte die PHP-Plattform erweitert werden durch die Protokolle:

Webfinger

Ein offenes Protokoll um Informationen über Benutzer und andere Objekte zur Verfügung zu stellen.

Interessant um Informationen über einen Benutzer zu erfahren und die Standardisierung von APIs.

PubSubHubbub

Ein offenes Protokoll um dezentrale Public-Subscribe prozesse zu implementieren. Dies könnte implementiert werden um die Replikation-Prozesse beim Speichern besser zu handlen.

Plattform

Um das Konzept besser zu "vermarkten" wäre eine vollständige Implementierung der Plattform von Vorteil. Es könnte die Vorteile des Konzeptes zeigen und es vollständig unterstützen.

UI Features wären:

- das teilen von Inhalten
- das editieren von Inhalten inklusive einigen Editoren
- das suchen von Verbindungen in der Oberfläche mittels Webfinger

Also alles zusammenzufügen.

Anschließend das Ausformulieren einer Spezifikation um es anderen zu ermöglichen das Konzept in einer anderen Programmiersprache umzusetzen.

THE END